Algoritmos e Estruturas de Dados I

CENTRO UNIVERSITÁRIO UNISENAC – CAMPUS PELOTAS
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
PROF. EDÉCIO FERNANDO IEPSEN

Estrutura de dados

Pinto, Rafael Albuquerque

1 O que é estrutura de dados?

Na computação, a estrutura de dados consiste no modo de armazenamento e organização de dados em um computador. Quando estamos nos referindo ao armazenamento de dados, existe uma infinidade de opções nas quais o dado poderá ser armazenado de acordo com o seu tipo e base de dados. Os dados podem ser armazenados em pastas, arquivos de texto, planilhas ou até mesmo em um banco de dados instalado localmente ou em nuvem.

Os dados armazenados não consistem somente no local no qual são armazenados, mas nas relações entre eles. A organização dos dados é tão importante quanto o modo no qual são armazenamos, pois a correta escolha de sua categoria irá acarretar na *performance* ou na velocidade que o computador, ou usuário, irá demorar para encontrá-los. Logo, a correta escolha de como devemos armazenar e organizar os dados deverá levar em consideração o modo no qual eles serão utilizados no futuro.

No exemplo a seguir, você poderá identificar como as nossas ações diárias estão ligadas diretamente aos conceitos de dados e às suas respectivas estruturas, bem como o seu efeito se não utilizadas de forma adequada.

Exemplo

Estruturas de Dados: Listas

Uma lista é uma variável que pode possuir vários elementos. Pode-se acessar individualmente cada elemento a partir de um índice que acompanha a variável. Em Python, uma lista é representada como uma sequência de objetos separados por vírgula e **dentro de colchetes []**.

```
idade = []              # cria uma lista vazia
idade = [20, 15, 30]    # cria uma lista com elementos
```

A lista inicia pelo índice 0.

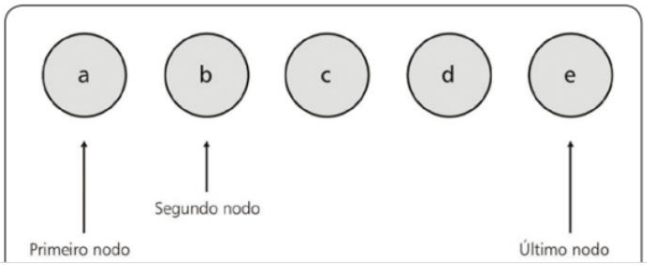
Estrutura de dados

Pinto, Rafael Albuquerque

1 O que são listas?

Na computação, uma lista pode ser definida como um conjunto de elementos do mesmo tipo, agrupados e identificados por um identificador único e separados entre si em "caixas", chamados de nodos, que ocupam um endereço específico na memória. O relacionamento entre os nodos é definido por sua posição em relação aos demais nodos, assim como pessoas em uma fila, porém indexados na memória do computador.

Toda lista apresenta um nodo inicial, o primeiro elemento da lista. A partir deste seguirá uma sequência de nodos conforme uma ordem predefinida pelo programador, assim como uma fila de banco. Todos os nodos de uma lista têm, geralmente, o mesmo tipo de dado, podendo ser do tipo primitivo, como um inteiro ou abstrato, criado pelo programador. Na Figura 1 é possível verificar de forma ilustrada um exemplo de nodo com a sua respectiva posição.



Funções para manipulação de listas

```
idade.append(12)      # acrescenta um elemento ao vetor idade
idade.pop()           # retira um elemento do vetor. Sem parâmetros, retira o último.
                      # Ou então, indica-se o número do elemento a ser removido
idade.insert(0, 5)    # indica o local da inserção (índice, conteúdo)
idade.remove(12)      # remove um elemento pelo conteúdo (se existir)
idade = range(5)      # cria um vetor com os valores [0, 1, 2, 3, 4]
len(idade)            # retorna o tamanho (número de elementos) do vetor
max(idade)            # maior valor
min(idade)            # menor valor
sum(idade)            # soma os elementos do vetor
12 in idade           # verifica se existe
idade.count(12)       # conta o número de ocorrências
idade.sort()          # classifica os elementos do vetor
idade.reverse()       # inverte a ordem dos elementos da lista
```

Métodos sort() x sorted(), reverse() x reversed()

- sort() e reverse() modificam a lista original
- sorted() e reversed() retornam uma nova lista

Localizar item na lista: método .index()

A sintaxe do `index()` método fica assim:

```
my_list.index(item, start, end)
```

Vamos decompô-lo:

- `my_list` é o nome da lista que você está pesquisando.
- `.index()` é o método de pesquisa que usa três parâmetros. Um parâmetro é obrigatório e os outros dois são opcionais.
- `item` é o parâmetro necessário. É o elemento cujo índice você está procurando.
- `start` é o primeiro parâmetro opcional. É o índice a partir do qual você iniciará sua pesquisa.
- `end` o segundo parâmetro opcional. É o índice onde você terminará sua pesquisa.

Se você tentar pesquisar um item, mas não houver correspondência na lista que está pesquisando, o Python lançará um erro como valor de retorno - especificamente, retornará um arquivo `ValueError`.

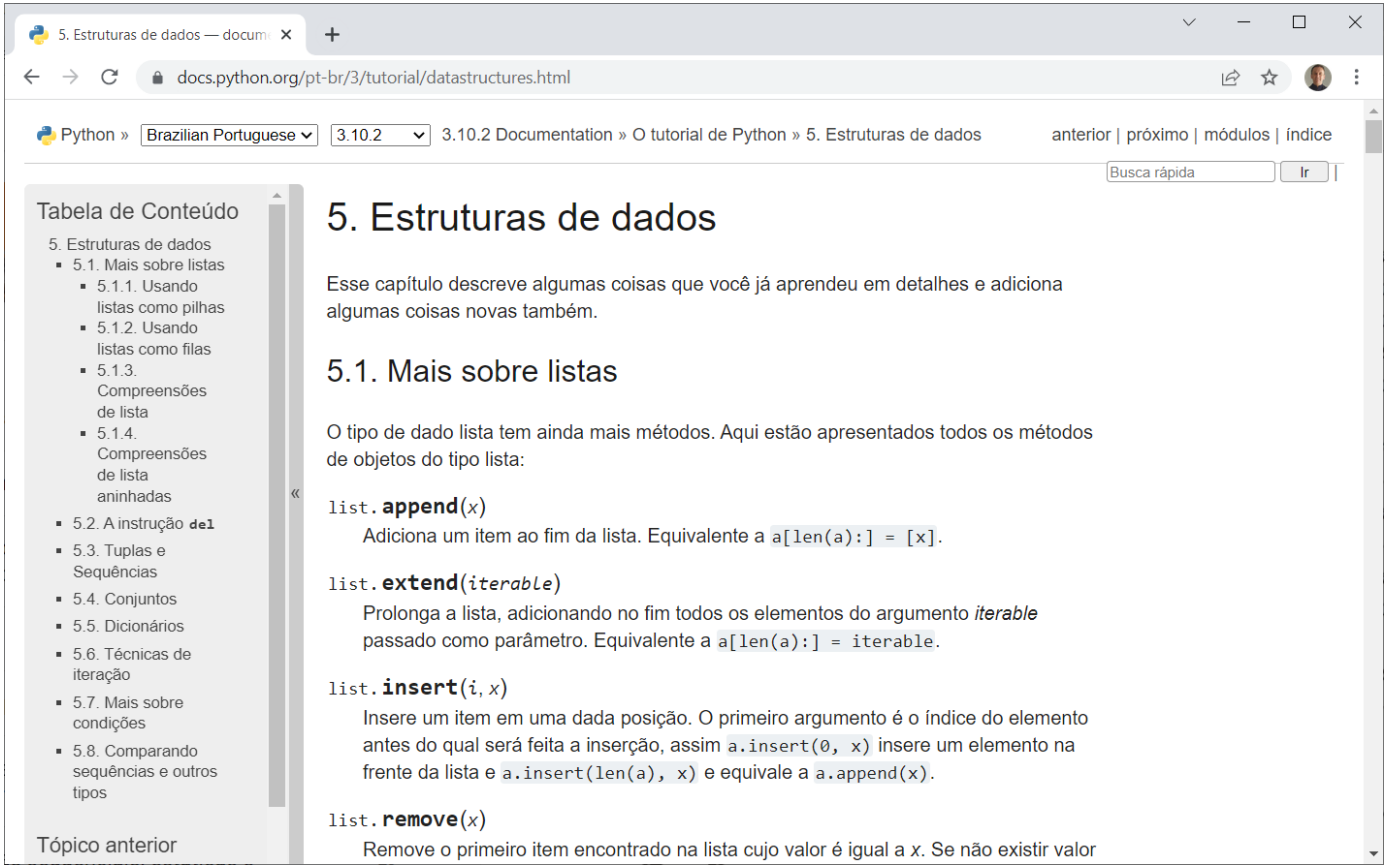
Isso significa que o item que você está procurando não existe na lista.

Uma maneira de evitar que isso aconteça é agrupar a chamada do `index()` método em um `try/except` bloco.

Se o valor não existir, haverá uma mensagem para o console dizendo que ele não está armazenado na lista e, portanto, não existe.

```
programming_languages = ["JavaScript","Python","Java","Python","C++","Python"]

try:
    print(programming_languages.index("React"))
except ValueError:
    print("That item does not exist")
```



Documentação: https://docs.python.org/pt-br/3/tutorial/datastructures.html

5.3. Tuplas e Sequências

Vimos que listas e strings têm muitas propriedades em comum, como indexação e operações de fatiamento. Elas são dois exemplos de *sequências* (veja Tipos sequências — list, tuple, range). Como Python é uma linguagem em evolução, outros tipos de sequências podem ser adicionados. Existe ainda um outro tipo de sequência padrão na linguagem: a *tupla*.

Uma tupla consiste em uma sequência de valores separados por vírgulas, por exemplo:

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
>>> u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> # Tuples are immutable:
>>> t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
>>> v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```

Como você pode ver no trecho acima, na saída do console as tuplas são sempre envolvidas por parênteses, assim tuplas aninhadas podem ser lidas corretamente. Na criação, tuplas podem ser envolvidas ou não por parênteses, desde que o contexto não exija os parênteses (como no caso da tupla dentro de uma expressão maior). Não é possível atribuir itens individuais de uma tupla, contudo é possível criar tuplas que contenham objetos mutáveis, como listas.

Apesar de tuplas serem similares a listas, elas são frequentemente utilizadas em situações diferentes e com propósitos distintos. Tuplas são imutáveis, e usualmente contém uma sequência heterogênea de elementos que são acessados via desempacotamento (ver a seguir nessa seção) ou índice (ou mesmo por um atributo no caso de `namedtuples`). Listas são mutáveis, e seus elementos geralmente são homogêneos e são acessados iterando sobre a lista.

Usando `zip()` em Python

A função do Python `zip()` é definida como `zip(*iterables)`. A função recebe iteráveis como argumentos e retorna um **iterador**. Este iterador gera uma série de tuplas contendo elementos de cada iterável. `zip()` pode aceitar qualquer tipo de iterável, como arquivos, listas, tuplas, dicionários, conjuntos e assim por diante.

Passando `n` argumentos

Se você usar `zip()` with `n` argumentos, então a função retornará um iterador que gera tuplas de comprimento `n`. Para ver isso em ação, dê uma olhada no seguinte bloco de código:

Pitão

```
>>> numbers = [1, 2, 3]
>>> letters = ['a', 'b', 'c']
>>> zipped = zip(numbers, letters)
>>> zipped  # Holds an iterator object
<zip object at 0x7fa4831153c8>
>>> type(zipped)
<class 'zip'>
>>> list(zipped)
[(1, 'a'), (2, 'b'), (3, 'c')]
```

https://realpython.com/python-zip-function/

Ordenar 2 listas

```
>>> letras = ['b', 'a', 'c']
>>> numeros = [3, 2, 1]
>>> juntas_ordenadas = sorted(zip(letras, numeros))
>>> juntas_ordenadas
[('a', 2), ('b', 3), ('c', 1)]
```

```
>>> for item in juntas_ordenadas:
...     print(item)
...
('a', 2)
('b', 3)
('c', 1)
```

Módulos (funções definidas pelo usuário)

Permite dividir um programa em pequenos trechos de código, onde cada um tem uma função bem definida.

Além da facilidade em lidar com trechos menores, pode-se também fazer uso da reutilização de código, já que estes trechos devem ser bem independentes.

As funções são uma forma de estruturação de programas quase universal. Em termos simples, uma função serve para agrupar um conjunto de instruções, de modo que elas possam ser executadas mais de uma vez em um programa.

Funções em Python

```
def cad_usuario():          # cria uma função
  print('...')


cad_usuario()               # chama a função
```

Funções com passagem de parâmetros

```
def ver_numero(num):     # num é passado por parâmetro
  if num % 2 == 0:
    print(f'{num} é par')
  else:
    print(f'{num} é ímpar')


ver_numero(5)             # chama a função passando 5
```

Funções com retorno de valor

```
def ver_numero(num):
  if num % 2 == 0:
    return f'{num} é par'        # retorno da função
  else:
    return f'{num} é ímpar'      # retorno da função

resp = ver_numero(5)        # retorno é atribuído a resp
```

Funções com retorno de valor

```
def ver_numero(num):
  if num % 2 == 0:
    tipo = f'{num} é par'
  else:
    tipo = f'{num} é ímpar'
  return tipo                # retorno da função

resp = ver_numero(5)         # retorno é atribuído a resp
```

Funções com vários parâmetros

```
def titulo(texto, traco):
  print()
  print(texto)
  print(traco*40)

titulo("Algoritmos e Estrutura de Dados", "=")
```

Funções com parâmetros com valores *default*

```
def titulo(texto, traco="-"):
  print()
  print(texto)
  print(traco*40)

titulo("Algoritmos e Estrutura de Dados", "=")
titulo("Programação Web")
```

Passagem de parâmetros nomeados

```
def titulo(texto, traco="-", num=40):
  print()
  print(texto)
  print(traco*num)

titulo("Aula 2")                    # parâmetros posicionais
titulo(num=20, texto="Aula 2")      # parâmetros nomeados
```

Exercícios – Manipulação de Listas

1. Elaborar um programa que exiba inicialmente o menu inicial apresentado a seguir:

```
1. Incluir Conta
2. Listar Contas
3. Listar Contas em Ordem
4. Pesquisar Conta
5. Excluir Conta
6. Finalizar
```

O programa deve ler descrição e valor de cada conta, armazenando os dados em 2 listas distintas.